Control de versiones con Git

aplicado a la investigación científica

Taller práctico — Astrojournal

Jennifer Grisales Casadiegos – Holman Quintero Salazar

Depto. de Astronomía — Universidad de Guanajuato · 2026

Agenda

01	¿Qué es Git y cómo funciona?	10 min
02	¿Por qué Git en ciencia?	5 min
03	Manos a la obra: del init al commit	20 min
04	Conectar con GitHub	10 min
05	Arquitectura de proyecto científico	10 min
06	Push final y cierre	5 min

01

¿Qué es Git y cómo funciona?

Un sistema de control de versiones distribuido

¿Qué es Git?

Git es un Version Control System (VCS)

Un VCS es un software que registra y administra cambios a archivos en el tiempo.

Permite:

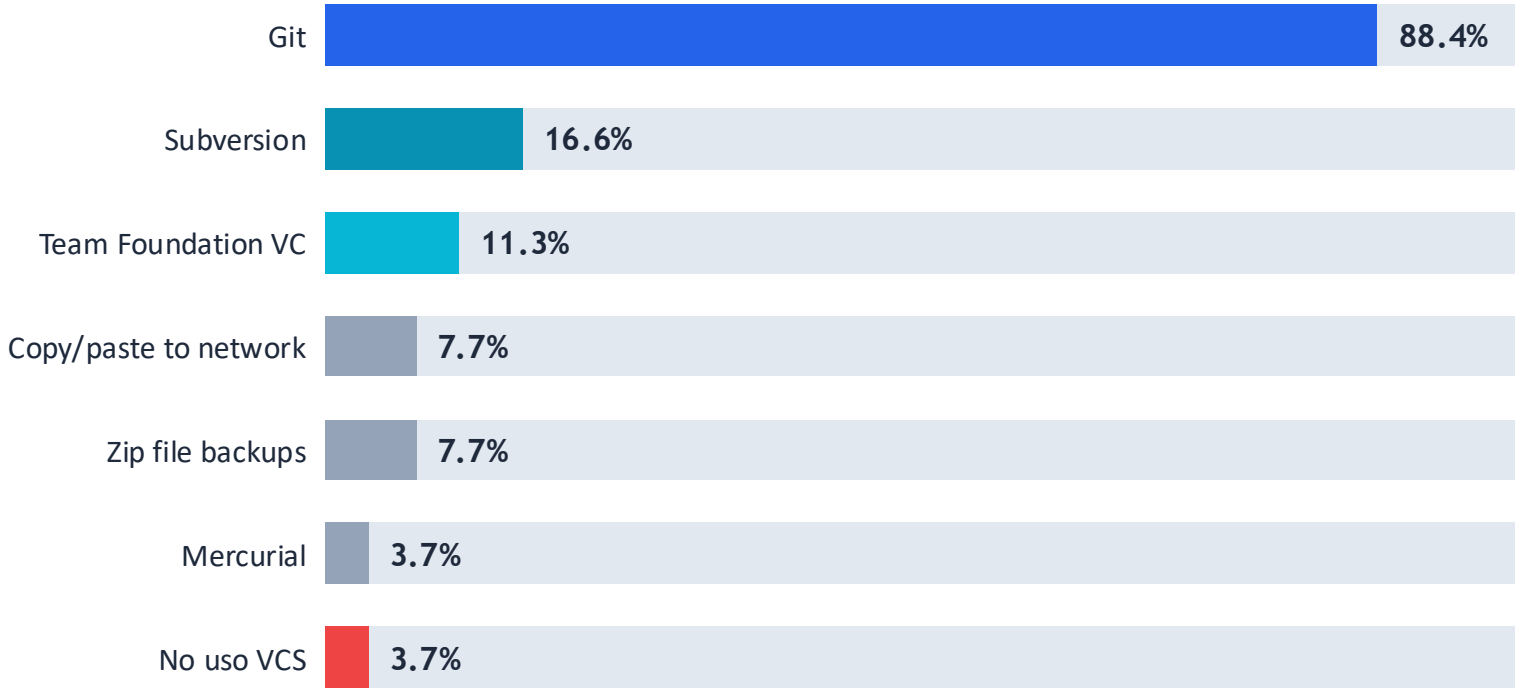
- Revisar versiones antiguas de archivos
- Comparar cambios entre versiones
- Deshacer cambios de forma controlada
- Colaborar sin sobrescribir el trabajo de otros

git-scm.com

Creado por Linus Torvalds en 2005 para gestionar el desarrollo del kernel de Linux.

Git es el estándar

Encuesta de Stack Overflow (2018) sobre VCS de preferencia



Después de estos resultados, Stack Overflow dejó de incluir esta pregunta en la encuesta.

¿Para qué nos sirve Git?

01 Registrar cambios a través de múltiples archivos

02 Comparar versiones de un proyecto

03 "Viajar en el tiempo" a versiones anteriores

04 Regresar a una versión anterior si algo se rompe

05 Colaborar y compartir cambios entre personas

06 Combinar cambios de distintas fuentes (merge)

La historia detrás de Git

2002

Linux + BitKeeper

El kernel de Linux adopta BitKeeper (propietario) como sistema de control de versiones.

2005

La ruptura

BitKeeper revoca la licencia gratuita. La comunidad Linux se queda sin herramienta.

Abr 2005

Torvalds crea Git

Linus Torvalds escribe Git en 10 días. Objetivos: velocidad, diseño distribuido, integridad de datos.

2008

GitHub

Se lanza GitHub. Git pasa de ser una herramienta de kernel hackers a estándar global.

Hoy

Estándar universal

Git es el VCS dominante en ciencia, industria y open source. >100M de repositorios en GitHub.

"Hablar es barato, muéstrame el código"

"I'm an egotistical bastard, and I name all my projects after myself. First Linux, now Git."
— Linus Torvalds, 2005

¿Quién usa Git?

No solo programadores.

Ingenieros y programadores

Tech-adjacent roles (DevOps, QA, data)

Gobiernos e instituciones públicas

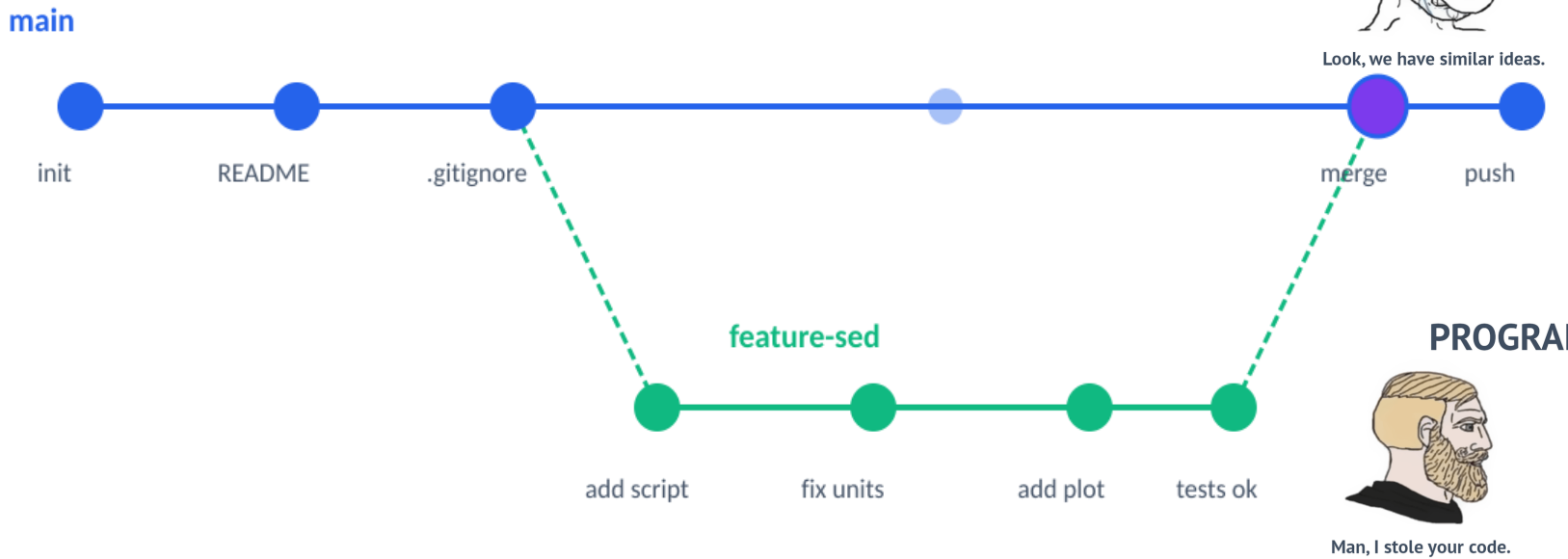
Científicos e investigadores

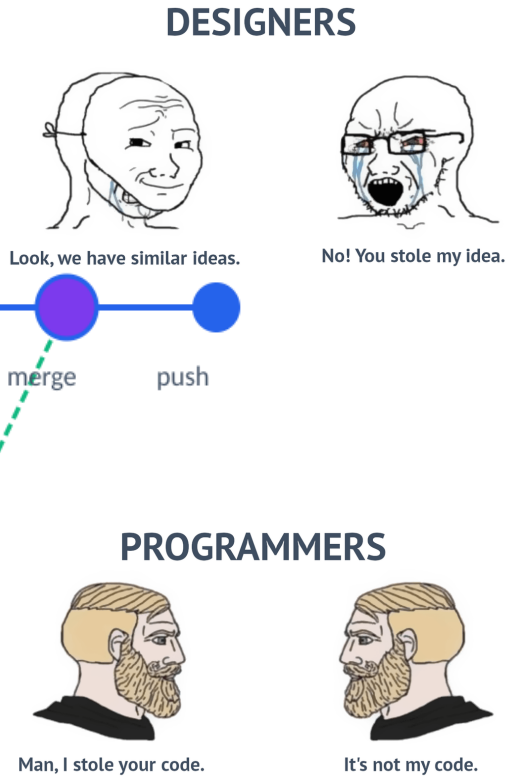
Escritores y documentalistas

Músicos, diseñadores, etc.

Cualquier persona que necesite rastrear cambios en archivos de texto de forma controlada.

Branches: trabajo en paralelo





La rama feature no afecta main.

Cuando está lista, se integra con git merge.

Dos personas pueden trabajar en paralelo

sin romper el código del otro.

Git ≠ GitHub

Git

Software de control de versiones que corre localmente en tu máquina.

No necesitas cuenta.
No necesitas internet.
Puedes usar Git sin tocar GitHub jamás.

GitHub

Servicio en la nube que aloja repositorios Git y facilita la colaboración.

Necesitas cuenta.
Necesitas internet.
Alternativas: GitLab, Bitbucket, Codeberg.

Git es el motor. GitHub es el garage donde lo estacionas y lo compartes.

Git: sistema de control de versiones distribuido

Local

Cada copia del repositorio es completa. Trabajas offline, sin depender de un servidor.

Distribuido

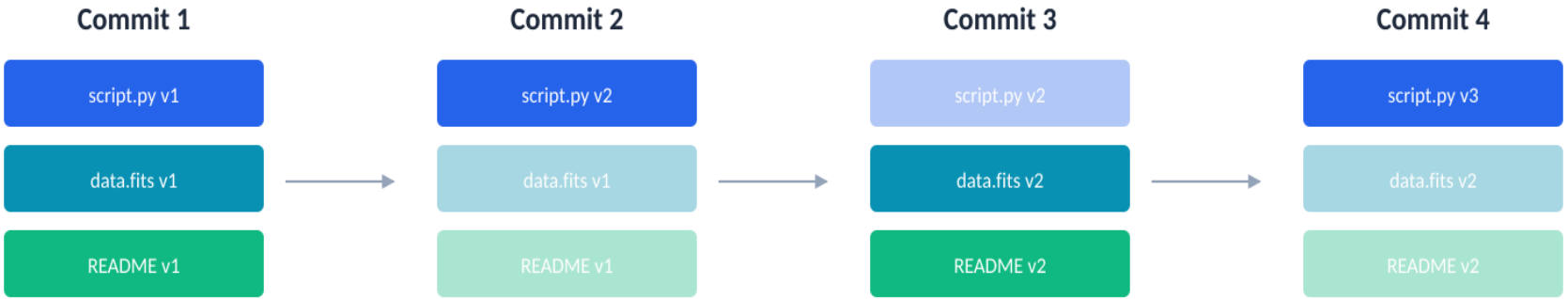
No hay un solo punto de fallo. Cada colaborador tiene el historial completo del proyecto.

No-lineal

Las ramas (branches) permiten desarrollar ideas en paralelo sin romper lo que ya funciona.

¿Cómo guarda Git los archivos?

Snapshots, no diferencias



↑ Sin cambios = Git guarda solo una referencia, no una copia

Archivo modificado (nuevo snapshot)

Sin cambios (referencia al snapshot anterior)

Cada commit es una fotografía completa del estado del proyecto, no un diff parcial.

Las tres áreas de Git

02

¿Por qué Git en investigación científica?

Reproducibilidad, trazabilidad, colaboración

Git resuelve problemas reales de tu flujo de trabajo

Sin Git

```
script_final_v3_ESTE_SÍ.py
script_final_v3_corregido.py
script_BACKUP_ene2026.py
```

No sabes qué cambió entre versiones.
No puedes volver atrás con control.
No puedes colaborar sin sobrescribir.
No puedes demostrar trazabilidad.

Con Git

```
git log --oneline
a3f2c1d  Fix SED normalization
7b8e4f0  Add H26α PV diagram
```

Historial completo de cada cambio.
Puedes revertir cualquier decisión.
Colaboración sin conflictos destructivos.
Reproducibilidad demostrable (FAIR).

Dos escenarios en astrofísica

Investigación

Scripts de análisis, notebooks, figuras para papers. No se publica como paquete, pero necesita trazabilidad y reproducibilidad.

Ej: CASA, astropy, matplotlib, notebooks de escuelas.

Software publicable

Paquete Python distribuible, afiliado a Astropy o publicado en PyPI/JOSS. Requiere tests, CI, documentación.

Ej: sf3dmodels, spectral-cube, picca, pyFADO, Cobaya.

Un proyecto bien estructurado puede evolucionar de Nivel 1 → Nivel 2 sin reescribir.

03

Manos a la obra

Del git init al primer commit — todos juntos

Paso 1

Verificar instalación y configurar identidad

```
$ git --version
$ git config --global user.name "Tu Nombre"
$ git config --global user.email "tu@email.com"
```

Cada commit lleva tu nombre y email. Esta configuración es global: una sola vez por máquina.

Ejercicio

Verifica que git --version devuelve 2.x.x y configura tu identidad.

Paso 2

Crear el proyecto e inicializar Git

```
$ mkdir mi_proyecto_astro
$ cd mi_proyecto_astro
$ git init
```

git init crea la carpeta oculta .git/ con toda la base de datos de versiones. Tu directorio ahora es un repositorio.

Ejercicio

Ejecuta ls -la y verifica que existe .git/

Paso 3

Primer archivo y primer commit

```
$ echo "# Mi Proyecto Astro" > README.md
$ git status
$ git add README.md
$ git commit -m "Inicio del proyecto"
```

git status → estado actual. git add → staging. git commit → snapshot permanente en el historial.

Ejercicio

Crea el README.md, revisa status antes y después del add, haz tu primer commit.

Paso 4

Modificar, ver cambios, commitear

```
  (edita README.md: agrega descripción)
$ git diff
$ git add README.md
$ git commit -m "Agrega descripción al README"
$ git log --oneline
```

git diff muestra qué líneas cambiaron. git log muestra el historial. Cada commit tiene un hash único (ej: a3f2c1d).

Ejercicio

Edita README, revisa diff, commit, y observa los 2 commits en git log.

Paso 5

Crear y usar un .gitignore

```
$ touch .gitignore
  (agrega: *.fits  *.pyc  __pycache__/  data/raw/)
$ git add .gitignore
$ git commit -m "Agrega .gitignore"
```

.gitignore dice a Git qué NO rastrear. Archivos FITS pesados, cachés de Python y datos crudos no deben ir al repositorio.

Ejercicio

Crea tu .gitignore con entradas para astrofísica. Haz commit.

Paso 6

Branches: trabajar en paralelo

```
$ git branch nueva-feature
$ git checkout nueva-feature
$ touch analisis.py
$ git add analisis.py
$ git commit -m "Inicia script de análisis"
$ git checkout main
$ git merge nueva-feature
```

Las ramas permiten experimentar sin afectar main. Cuando la feature está lista, se integra con merge.

Ejercicio

Crea una rama, añade un archivo, commit, vuelve a main, fusiona.

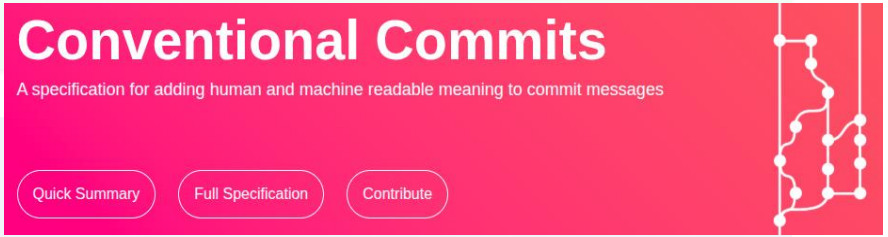
Tips para hacer commits

✗ Mal commit

```
Tengo hambre.exe
;klasldasjak
20/12/23
asdfghjkl
Me debes 100 pesos
```

✓ Buen commit

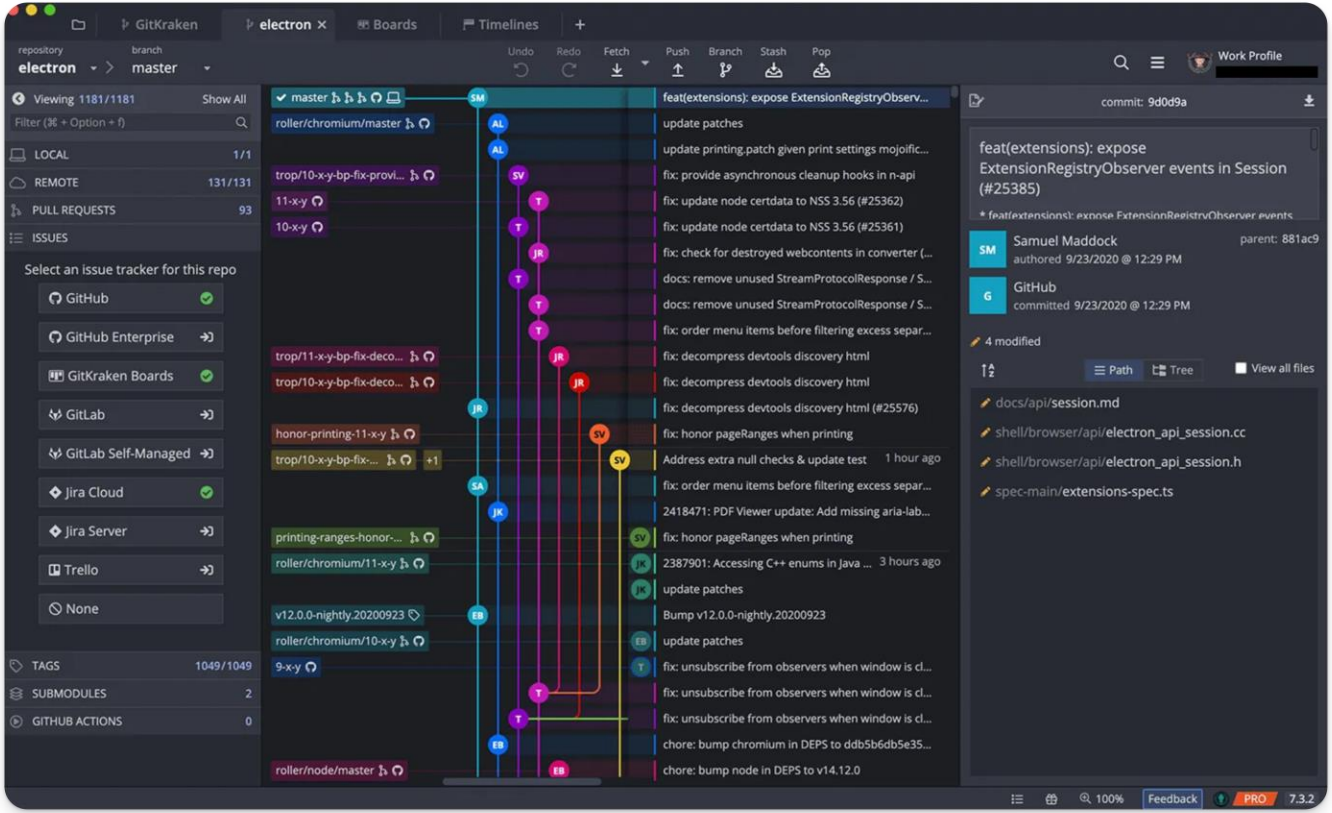
```
feat: add SED fitting module
fix: correct flux unit conversion
docs: update README with installation
refactor: split analysis into functions
data: add processed continuum maps
```



Conventional Commits — buenas prácticas

Formato: `tipo: descripción corta en imperativo`
Tipos comunes: feat (nueva funcionalidad) · fix (corrección) · docs (documentación) · refactor · data · test
Ref: conventionalcommits.org

GitKraken


GitKraken
electron
Boards
Timelines
repository
electron
branch
master
Undo
Redo
Fetch
Push
Branch
Stash
Pop
Work Profile
Viewing 1181/1181
Show All
Filter (⌘ + Option + f)
LOCAL 1/1
REMOTE 131/131
PULL REQUESTS 93
ISSUES
Select an issue tracker for this repo
GitHub
GitHub Enterprise
GitKraken Boards
GitLab
GitLab Self-Managed
Jira Cloud
Jira Server
Trello
None
TAGS 1049/1049
SUBMODULES 2
GITHUB ACTIONS 0
master
roller/chromium/master
trop/10-x-y-bp-fix-provi...
11-x-y
10-x-y
trop/11-x-y-bp-fix-deco...
trop/10-x-y-bp-fix-deco...
honor-printing-11-x-y
trop/10-x-y-bp-fix-...
+1
printing-ranges-honor-...
roller/chromium/11-x-y
v12.0.0-nightly.20200923
roller/chromium/10-x-y
9-x-y
roller/node/master
feat(extensions): expose ExtensionRegistryObserv...
update patches
update printing.patch given print settings mojofic...
fix: provide asynchronous cleanup hooks in n-api
fix: update node certdata to NSS 3.56 (#25362)
fix: update node certdata to NSS 3.56 (#25361)
fix: check for destroyed webcontents in converter (...
docs: remove unused StreamProtocolResponse / S...
docs: remove unused StreamProtocolResponse / S...
fix: order menu items before filtering excess separ...
fix: decompress devtools discovery html
fix: decompress devtools discovery html
fix: decompress devtools discovery html (#25576)
fix: honor pageRanges when printing
Address extra null checks & update test 1 hour ago
fix: order menu items before filtering excess separ...
2418471: PDF Viewer update: Add missing aria-lab...
fix: honor pageRanges when printing
2387901: Accessing C++ enums in Java ... 3 hours ago
update patches
Bump v12.0.0-nightly.20200923
update patches
fix: unsubscribe from observers when window is cl...
fix: unsubscribe from observers when window is cl...
fix: unsubscribe from observers when window is cl...
chore: bump chromium in DEPS to ddb5b6db5e35...
chore: bump node in DEPS to v14.12.0
commit: 9d0d9a
feat(extensions): expose ExtensionRegistryObserver events in Session (#25385)
Samuel Maddock
parent: 881ac9
authored 9/23/2020 @ 12:29 PM
GitHub
committed 9/23/2020 @ 12:29 PM
4 modified
Path
Tree
View all files
docs/api/session.md
shell/browser/api/electron_api_session.cc
shell/browser/api/electron_api_session.h
spec-main/extensions-spec.ts
100%
Feedback
PRO
7.3.2

04

Conectar con GitHub

Tu repositorio en la nube — SSH authentication

Conectar repositorio local ↔ GitHub

A. Generar clave SSH

```
$ ssh-keygen -t ed25519 -C "tu@email.com"
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_ed25519
$ cat ~/.ssh/id_ed25519.pub    # copiar esta salida
```

B. Agregar clave en GitHub → Settings → SSH and GPG Keys → New SSH Key

```
$ ssh -T git@github.com    # verificar conexión
```

C. Crear repo vacío en GitHub y conectar

```
$ git remote add origin git@github.com:usuario/mi_proyecto_astro.git
$ git branch -M main
$ git push -u origin main
```

Guía completa: github.com/jennifergc/git_guide

05

Arquitectura de proyecto

Nivel 1: Investigación · Nivel 2: Paquete Astropy

Nivel 1 — Proyecto de investigación reproducible

```
mi_proyecto/
├── README.md
├── LICENSE
├── .gitignore
├── environment.yml
├── data/
│   ├── raw/            ← .gitignore
│   └── processed/
├── notebooks/
├── src/mi_proyecto/
│   └── __init__.py
├── scripts/
├── results/
│   ├── figures/
│   └── tables/
└── docs/
```

README.md

Qué es el proyecto, dependencias, cómo replicar. Primer archivo que lee cualquier persona.

.gitignore

FITS (>100 MB) NO van a Git. Datos crudos nunca se versionan.

environment.yml

conda env export. Cualquiera puede recrear tu entorno exacto.

src/mi_proyecto/

Funciones reutilizables entre notebooks. El __init__.py facilita la transición a Nivel 2.

Ejercicio

Construyamos la estructura Nivel 1

```
# Desde mi_proyecto_astro/
$ mkdir -p data/raw data/processed
$ mkdir -p notebooks scripts
$ mkdir -p src/mi_proyecto_astro
$ touch src/mi_proyecto_astro/__init__.py
$ mkdir -p results/figures results/tables
$ mkdir docs

# .gitignore para astrofísica
$ echo "*.fits" >> .gitignore
$ echo "*.pyc" >> .gitignore
$ echo "__pycache__/" >> .gitignore
$ echo "data/raw/" >> .gitignore
$ echo ".ipynb_checkpoints/" >> .gitignore

$ git add -A
$ git commit -m "Estructura inicial del proyecto"
```

Nivel 2 — Paquete afiliado a Astropy

¿Qué se agrega al Nivel 1?

pyproject.toml

Reemplaza setup.py. Estándar PEP 517/518. Define nombre, versión, dependencias y metadatos.

tests/ con pytest

Tests unitarios obligatorios para afiliación Astropy. Garantizan que el código funciona al actualizar deps.

CI (GitHub Actions)

Integración continua: tests automáticos en cada push. Se configura en .github/workflows/.

Docs con Sphinx

Documentación formal con autoAPI. No opcional para un paquete público. Se publica en ReadTheDocs.

Ref: packaging-guide.openastronomy.org — guía oficial vigente (reemplaza astropy/package-template, archivado 2023)

Registra tu código: hazlo citable y permanente

Después de publicar tu repositorio, el siguiente paso es que sea citable y archivable.

ASCL

ascl.net

Astrophysics Source Code Library

Qué es: Registro curado de software usado en papers revisados por pares. +3,500 códigos indexados.

Qué registras: Tu código como entrada en el catálogo. Apunta a tu repo en GitHub.

Qué obtienes: Un ascl ID citable (ej: ascl:2306.025). Indexado por ADS y Web of Science.

Para qué sirve: VISIBILIDAD. Que otros astrofísicos encuentren y citen tu trabajo desde ADS.

Zenodo

zenodo.org

Archivo digital abierto — CERN / OpenAIRE

Qué es: Repositorio de archivo permanente operado por el CERN. Para cualquier output de investigación.

Qué registras: Un snapshot congelado de tu repo (ZIP). Se integra directo con GitHub: cada release → archivo automático.

Qué obtienes: Un DOI por cada versión (ej: 10.5281/zenodo.1234567). Citable en cualquier journal.

Para qué sirve: Preservación permanente. Tu código sobrevive aunque borres el repo de GitHub.

Git + GitHub → **Zenodo (DOI)** → **ASCL (descubribilidad ADS)** Son complementarios, no excluyentes.

06

Push final y cierre

Tu proyecto en GitHub, visible para el mundo

Push final: tu repositorio en GitHub

```
# Verificar que todo está commiteado
$ git status
# Push al remote
$ git push -u origin main
```

Antes de cerrar, verifica:

- ✔ Abre github.com/tu-usuario/mi_proyecto_astro
- ✔ ¿Se ve el README.md renderizado?
- ✔ ¿El .gitignore está presente?
- ✔ ¿La estructura de carpetas es correcta?
- ✔ ¿Los archivos .fits y data/raw/ NO aparecen?

Referencia rápida de comandos

Comando	Función
`git init`	Inicializa repositorio
`git status`	Estado actual del repo
`git add <archivo>`	Lleva archivos al staging
`git commit -m "msg"`	Crea snapshot con mensaje
`git log --oneline`	Historial compacto
`git diff`	Cambios no staged
`git branch <nombre>`	Crea rama nueva
`git checkout <rama>`	Cambia de rama
`git merge <rama>`	Fusiona rama en actual
`git remote add origin <url>`	Conecta con remoto
`git push -u origin main`	Sube commits al remoto
`git pull`	Descarga y fusiona del remoto
`git clone <url>`	Clona repositorio completo

Tu código merece la misma rigurosidad que tus datos.

Recursos

Git Guide: github.com/jennifergc/git_guide
OpenAstronomy Packaging: packaging-guide.openastronomy.org
The Turing Way: the-turing-way.netlify.app
Pro Git Book (gratis): git-scm.com/book/es

Jennifer Grisales & Holman Quintero · Depto. de Astronomía · UG · 2026